

# What is NLP?

Computers speak in binary (0 and 1) because of how they are built. Computers don't understand human language. In simple words, teaching human language to computers.

## Standard definition -

Natural language processing NLP is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. (Wikipedia)

## Why do we need NLP?

We need NLP to make computers/machines more accessible. For example, let's say someone doesn't know how to use an ATM; in that case, an NLP application that will guide its user step by step to perform certain functions will be very useful.

## NLP has many real-life applications like -

- Voice assistants - voice assistants like Siri, Alexa, Cortana, and google assistant are applications of natural language processing.
- Spam filtering and smart reply
- Recommendation systems - while building recommendation systems, we use NLP. Recommendation systems can be of movies, books, songs, etc.
- Sentiment analysis - Let's say we own an e-commerce website and we want to know what customers of some particular product feel about that product. We can do sentiment analysis on product reviews to see if customers are liking that product or not.
- Language translators - NLP is used in language translators.

## NLP Pipeline

To build any NLP application, there are some common steps that we follow. A set of steps to build an end-to-end NLP software is known as the NLP pipeline.

Steps in the NLP pipeline are -

- Data acquisition -

Let's say we are building an NLP application. The first thing we need to build an NLP application is data.

There are many methods to acquire the data, like:

Public datasets - We can take datasets from websites like Kaggle or some university websites where the dataset is publicly available.

Web scraping - Taking data from websites can be another option. For doing this there are python libraries like beautiful soup.

API to get data - many websites provide their APIs so that we can acquire their data easily. RapidAPI.com is one website that has a list of all the APIs in different domains.

- Text preprocessing -

The data we acquired in the previous step will not always be in the form we want. It will be raw data, and to make it usable, we have to do some data preprocessing.

The following processes are present in text preparation -

1. Lowercasing the text
2. Removing punctuations - punctuation marks like . , ? ! don't have any particular meaning, so we remove them.
3. Removing stop words - Stop words are words used in a sentence to make a meaningful sentence but those words do not have their own meaning. E.g. - a, the, is, are
4. Tokenization- Tokenization is classified into two types. Word tokenization and sentence tokenization

Word tokenization -

Let's say we have a sentence - I eat apple

And we tokenized this sentence, so we'll get an array with each element being a word from the sentence.

["I", "eat", "apple"]

Sentence tokenization -

It is the same as the previous process, but instead of a sentence, here we have a document, and tokenizing it will give us an array whose elements will be different sentences from that document.

5. Stemming - If we have "change", "changed" and "changing" in our data, we know that these 3 words have the same meaning, but machines don't know that, so we convert these 3 words into 1 base form, and this process is known as stemming.
6. POS tagging - we will see this step later.
7. Remove emojis
8. Spell check

- Feature engineering

We can't apply any algorithm on textual data. We have to convert words/sentences/paragraphs in input data to numbers in a meaningful way. (we can't just assign random numbers)

There are several methods to convert text into numbers -

1. Bag of words
2. TF-IDF
3. Word2Vec

Let's look at these methods one-by-one -

## 1. Bag of words -

Bag of words converts sentences into a vector.

Let's take an example to see how bag of words work

Say we have 3 sentences -

1. He is a good boy
2. She is a good girl
3. Boy and girl are good

After text preparation we will get something like -

1. good boy
2. good girl
3. boy girl good

Now let's create a matrix-like structure where rows are sentences in a data and columns are all the different words present in a data. -

	boy	girl	good
Sentence 1(good boy)	1	0	1
Sentence 2(good girl)	0	1	1
Sentence 3(boy girl good)	1	1	1

Now we got a vector for each sentence -

1. He is a good boy - [1,0,1]
2. She is a good girl - [0,1,1]
3. Boy and girl are good - [1,1,1]

In the bag of words no semantic meaning is captured which means in the bag of words it does not matter if we change the order of words in a sentence. But as we know in textual data order of words is very important.

## 2. TF-IDF

TF IDF stands for term frequency-inverse document frequency. It converts sentences into vectors.

Let's assume we have 1 corpus(D) which contains 3 sentences d1,d2,d3.

We calculate TF part as follows -

$$TF(t, d) = \frac{\text{number of occurrences of } t \text{ in } d}{\text{total number of words in } d}$$

$t$  – word,  $d$  – sentence

We calculate IDF part as follows -

$$IDF(t, D) = \log\left(\frac{\text{total number of sentences in } D}{\text{total number of sentences with } t \text{ in it}}\right)$$

And then we take their product to calculate TF-IDF

$$TF - IDF = TF * IDF$$

## 3.Word2Vec

Word2Vec is developed by Google engineers. It is a pre-trained model. It is trained on the Google News dataset. Word2Vec converts words into vectors. It captures the semantic meaning of the words. It converts the words into low-dimensional vectors as compared to a bag of words and TF-IDF. It gives us dense vectors.

Word2Vec will try to create features to identify each word.

For e.g - for a word 'human' we can say features can be height, weight,gender etc.

Let's take an example to understand how Word2Vec captures the semantic meaning -

Let's take 5 words - king, queen, man, woman, and monkey. Let's say the features Word2Vec created are gender, wealth, power, weight, and speak. (In Word2Vec, there is no way of knowing what features our model created; here, I am taking these 5 features by myself to make you understand how Word2Vec captures semantic meaning.)

	King	Queen	Man	Woman	Monkey
Gender	1	0	1	0	1
Wealth	1	1	0.3	0.3	0
Power	1	0.7	0.2	0.2	0
Weight	0.8	0.4	0.6	0.5	0.3
Speak	1	1	1	1	0

Here, the numbers I used for each feature will be determined by our model, but I have taken these numbers from my general knowledge just to understand the concept.

In the table above, the values of each feature vary from 0 to 1. For gender, the feature value is 0 if it is female and 1 if it is male. The same is true for the feature speak: if it can speak, the value is 1, otherwise it is 0. For the wealth, power, and weight features, the value varies from 0 to 1 depending on how wealthy, powerful, or heavy it is.

Now we have a vector for each word -

King = [1,1,1,0.8,1]  
Queen = [0,1,0.7,0.4,1]  
Man = [1,0.3,0.2,0.6,1]  
Women = [0,0.3,0.2,0.5,1]  
Monkey = [1,0,0,0.3,0]

Now we can do word math like - (king - man + woman) and we should get a vector similar to a queen.

Let's try -

King - man + woman = [1,1,1,0.8,1] - [1,0.3,0.2,0.6,1] + [0,0.3,0.2,0.5,1]  
= [0,1,1,0.7,1]  
≈ queen

We got [0,1,1,0.7,1] as a result which is similar to the queen's vector representation.

The underlying assumption of Word2Vec is that 2 words sharing similar contexts also share a similar meaning and consequently similar vector representation.

E.g. -

Let's say we have 2 sentences -

1. Delhi is the capital of India
2. Paris is the capital of France

The pair Delhi, India, and Paris, France are used in the same context. Therefore Word2Vec will be able to understand that the relationship between Delhi and India is similar to Paris and France.

## How does word2Vec find the weights of words?

Word2Vec takes a fake problem and in process of solving that problem, we get a vector for a word as a byproduct

There are 2 methods to determine the weights -

1. CBOW - continuous bag of words
2. Skip-gram

Here we will look at the CBOW method

Let's say we have a sentence -

Watch youtube videos for fun

Let's take a word window of 3. (it just means at 1 time we will take a set of 3 words)

Let's say we take watch youtube videos set.

So here we will take youtube as a target word and watch, videos as context words.

Slide the set of 3 words by 1 word and each time we will get new target words and its corresponding context words.

Now we can make training data like -

X(context)	Y(target)
Watch, videos	youtube
Youtube, for	videos

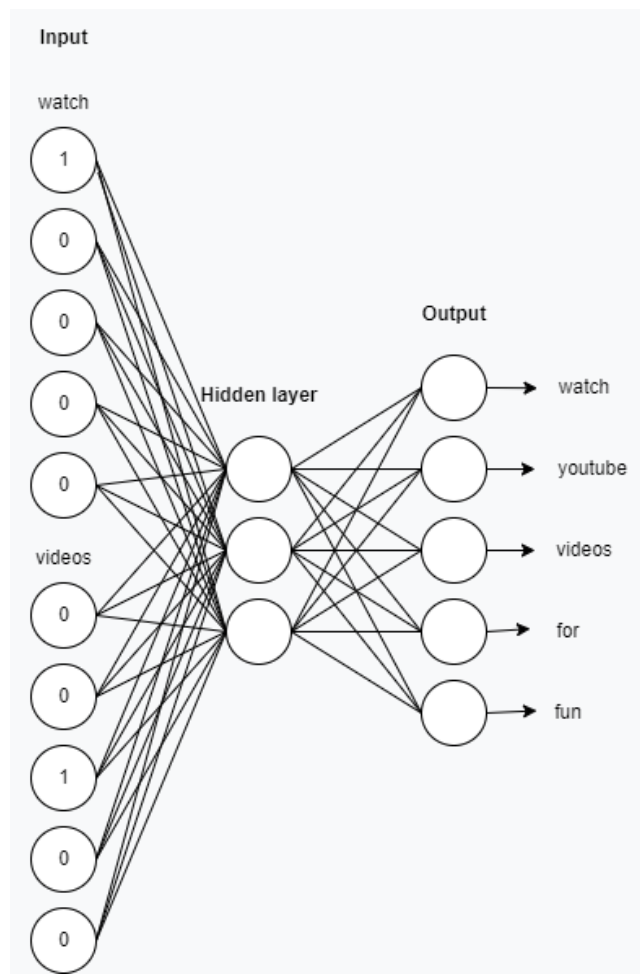
Videos, fun	for
-------------	-----

By using one hot encoding let's convert our sentence into vectors -

Watch - [1,0,0,0,0]  
 Youtube - [0,1,0,0,0]  
 Videos - [0,0,1,0,0]  
 For - [0,0,0,1,0]  
 Fun - [0,0,0,0,1]

We have training data, we will create a neural network which will have 2 inputs(context) and 1 output(target).

In this example I will be taking a hidden layer of 3 nodes.(number of nodes in hidden layer = number of features of vectors)



In the above diagram, I took "watch" and "videos" as input, and we are trying to guess the target, which is "youtube". We will do this for every row in the training data. After training the model, the weight matrix from the hidden layer to the output layer, which has a dimension of 3x5, will be our vectors for words.

It is a multiclass classification problem.

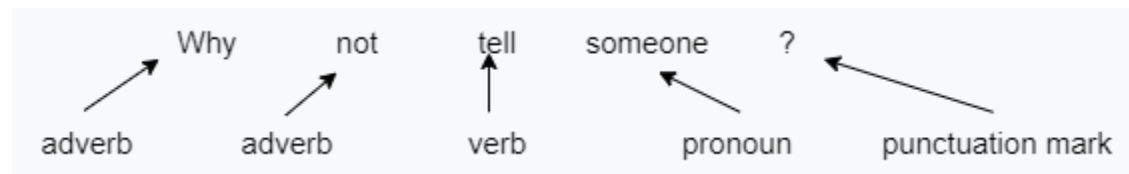
In the skip-gram method, we will try to guess the context by inputting the target word; everything else remains the same as CBOW.

## POS tagging

What is POS(part of speech) tagging?

A task of labelling each word in a sentence with its appropriate part of speech.

E.g. -



POS tagging is a data preprocessing step.

What are the applications of POS tagging?

1. Named entity recognition

It is the information retrieval process.

Let's say we have a sentence -

Rahul will go to Delhi on 30th March

Named entity recognition will use Rahul, Delhi and 30th March as output

2. Question-answering system

POS tagging is used as a data preprocessing step while making question-answering systems.

3. Word sense disambiguation

If one word is being used in two different sentences with different meanings, one can tell the difference



between them after POS tagging.

E.g. -

Let's say we have 2 sentences -

1. Rahul took a left turn
2. Rahul left the meeting

The word “left ” in both sentences has a different meaning. After doing POS tagging machines will also understand that both “left” are different.

#### 4. Chatbots

## How does POS tagging work?

→ Computer uses. Hidden Markov Model (HMM) algorithm for doing Pos tagging.

Let's understand how HMM works through example -

→ Let's say we have 4 sentences in our training data and all words of these 4 sentences are tagged with their appropriate part of speech.

(N)	(V)	(N)	
Nitish	loves	tesla	
(M)	(N)	(V)	(N)
Can	Nitish	google	tesla
(M)	(N)	(V)	(N)
will	Ankita	google	tesla
(N)	(V)	(N)	
Ankita	loves	Will	

N - noun

V - verb

M - modal auxiliary

Now we have to find the emission Probability for our training data.

Let's make a table whose rows are all the different words in our training data and whose columns are all types of Parts of speech present in training data.

In this table, we will write all the time that a particular word is used as a noun, pronoun verb etc.

	N (noun)	M (modal auxiliary)	V (verb)
nitish	2	0	0
loves	0	0	3
tesla	3	0	0
google	1	0	2
will	2	1	0
ankita	2	0	0
can	0	1	0

Now we will divide the noun column by the number of times nouns occurred in our training data (2+3+2+1+2=10). Do the same for other columns. And that will be our emission probabilities.

	N (noun)	M (modal auxiliary)	V (verb)
nitish	2/10	0	0
loves	0	0	3/5
tesla	3/10	0	0
google	1/10	0	2/5
will	2/10	1/2	0
ankita	2/10	0	0
can	0	1/2	0

→Emission probability is if we have a word the probability of that word being a particular part of speech.

→ Next thing we need is transition probability.

transition probability will tell us what is probability. of getting some part of speech next to another Part of speech.

E.g. - The probability of having a verb next to the noun. and all the possible combinations of Parts of speeches present in our dataset.

First, we have to add a start (S) and an end (E) in each sentence.

	N	V	N	
S	Nitish	loves	tesla	E

Do this for every sentence

Create the following table

	N	M	V	E (end)
S (start)	3	2	0	0
N	0	0	5	5
M	2	0	0	0
V	5	0	0	0

Here the value Of block (S, N) represents the number of times N (noun) occurs after S(start). and the same for all the remaining pairs.

Now divide each row item by its Sum and we will get transition probabilities.

	N	M	V	E (end)
--	---	---	---	---------

S (start)	3/5	2/5	0	0
N	0	0	5/10	5/10
M	2/2	0	0	0
V	5/5	0	0	0

→Now our hidden Markov model is ready.

Let's try to get parts of speeches of each word of the following sentence-

Will      will      google      tesla

Let's assume all the words are nouns (N).

Will      will      google      tesla  
S — N — N — N — N — E

We have all the probabilities like the probability of “will” being a noun and the probability of occurrence of a noun after Start and so on. Let's assign those probabilities.

Will      will      google      tesla  
S  $\frac{3}{5}$  N  $\frac{0}{2/10}$  N  $\frac{0}{2/10}$  N  $\frac{0}{1/10}$  N  $\frac{5/10}{3/10}$  E

Multiply all the probabilities.

$$\frac{3}{5} \cdot \frac{2}{10} \cdot 0 \cdot \frac{2}{10} \cdot 0 \cdot \frac{1}{10} \cdot 0 \cdot \frac{3}{10} \cdot \frac{5}{10} = 0$$

The probability of all the words being nouns is 0.

like this we will try all the combinations of parts of speech on all the words in a sentence and calculate the probability of that combination.

The combination with the highest probability is our final answer

Let's try to find the probability for the correct combination

	Will	will	google	tesla	
S	$\frac{2}{5}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	E
	M	N	V	N	
	$\frac{1}{2}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{10}$	

$$\frac{2}{5} \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{5} \cdot \frac{1}{2} \cdot \frac{2}{5} \cdot 1 \cdot \frac{1}{10} \cdot \frac{1}{2} = \frac{1}{2500}$$

And this will be the highest probability amongst all the probabilities of other combinations.

The result is -



Therefore these tags will be assigned to the words.

There is one problem with this method.

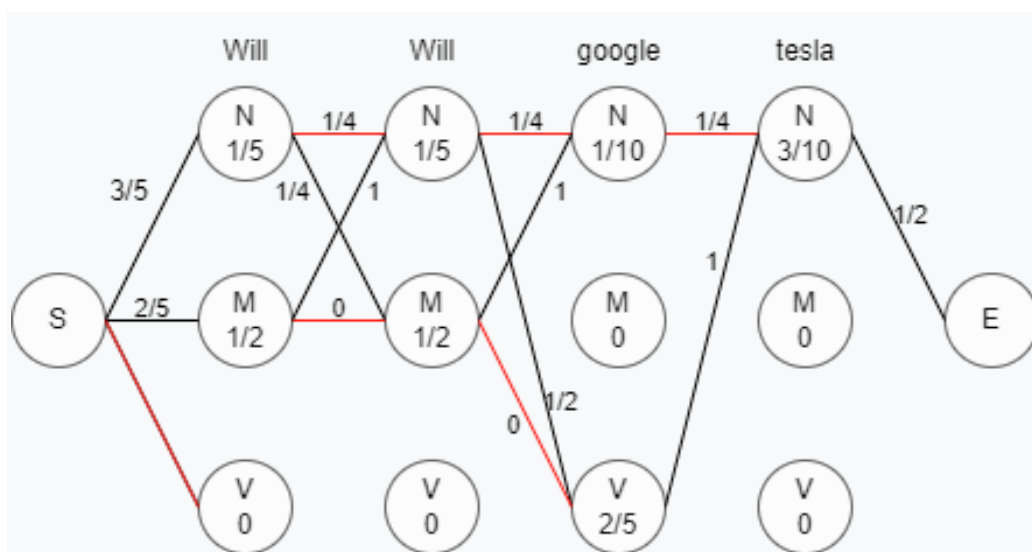
The example we took has 4 words and for each word 3 Part of speeches are Possible Therefore there are  $3^4 = 81$  combinations we have to check for 1 sentence. And. I took a simple example with 4 words and 3 POS. Imagine a 10 word sentence with 6 POS.

→To reduce this computational complexity we use the Viterbi Algorithm.

## Viterbi Algorithm.

To understand the Viterbi algorithm better I am taking the following transition probabilities.

	N	M	V	E (end)
S (start)	3/5	2/5	0	0
N	1/4	1/4	5/10	5/10
M	2/2	0	0	0
V	5/5	0	0	0



Each column represents. the probability of that word being N, M, or V.  
For example, the probability of google being a noun is 1/10.

These are emission probabilities.

Our goal is to find a path from start (S) to end (E) such that the product of transition and emission probabilities along that path is maximum.

Our 1st word is "will" So we will consider all paths from start (the previous word) and select the path with the maximum probability leading to "will" being a noun (I took a noun as an example; we will do this for other 2 also).

From S to "will" being a noun there is only 1 path. For now, we will consider this path.

From S to "will " being modal auxiliary, there is only 1 path. We will also consider this path.

The probability of "will" being a verb is 0. We will not consider the path from S to "will" being a verb. (Because we have to multiply all the probabilities in the final step, and we know that if we consider a path with at least one 0 value, we will almost certainly get a 0).

For a 2nd "will" being a noun it can come from 1st "will" as a noun or 1st "will" as a modal auxiliary. Another possibility is that 2nd "will" is a modal auxiliary for that also we have 2 routes that are 1st "will" being a noun or modal auxiliary.

However, we know that the probability of a modal auxiliary to a modal auxiliary transition is 0. Therefore, we don't have to consider that route.

For the 2nd "will" we have 3 routes (not considering verb routes because the probability of the 2nd "will" being a verb is 0).

3 routes are -

- 1st "will" to 2nd "will" N to N
- 1st "will" to 2nd "will" N to M
- 1st "will" to 2nd "will" M to N

We will now compute transition probabilities for these routes and only consider the route with the highest probability among these 3 options.

\* Now that we've reached the second "will," only one path can be taken based on the previous words' part of speech. We will eliminate the route with less probability.

e.g-

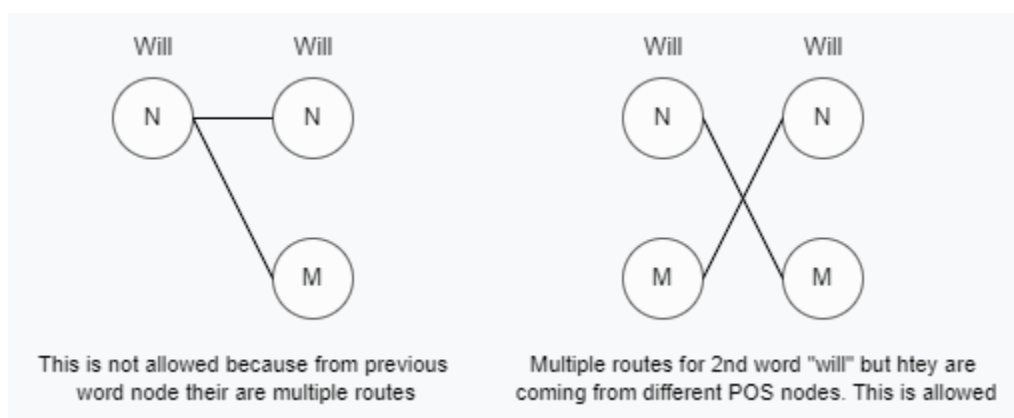


Fig. 1

Fig. 2

The condition in Fig. 1 is arising for the 2nd "will", which means we have to eliminate one of the paths.

We will calculate the product of emission and transition probabilities until the 2nd "will", and only keep the route with a greater probability.

For S-N-N route:

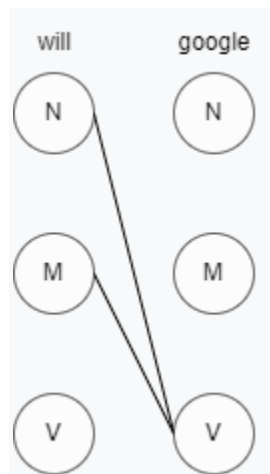
$$\frac{3}{5} \cdot \frac{1}{5} \cdot \frac{1}{4} \cdot \frac{1}{5} = \frac{3}{500}$$

For S-N-M route:

$$\frac{3}{5} \cdot \frac{1}{5} \cdot \frac{1}{4} \cdot \frac{1}{2} = \frac{3}{200}$$

Therefore, we will keep the route S-N-M.

In addition to \* following is also not allowed (means we have to eliminate route(s) in that case):



In this case, we have to eliminate one of the routes.

Do these steps for all the remaining words, and we will get the desired result.

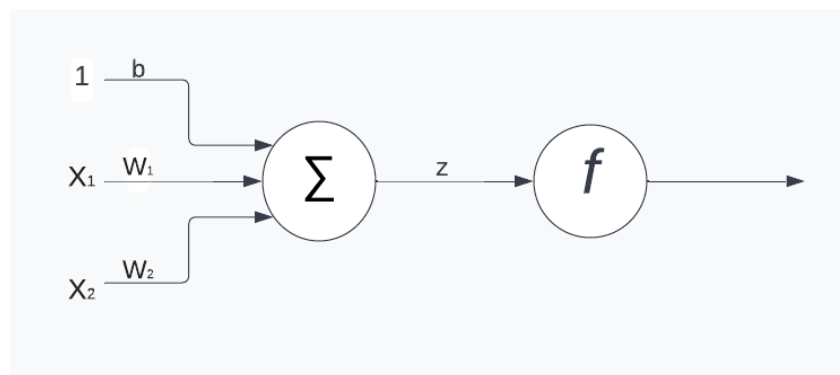


# Perceptron

What is a perceptron?

It is an algorithm. It is used for supervised machine learning. We can see the perceptron as a mathematical model or function.

Perceptron looks like this:



There are several things in this diagram.  $X_1$  and  $X_2$  are the inputs of the perceptron, and  $W_1$  and  $W_2$  are their respective weights.  $\Sigma$  is where all the inputs came from, and it does some math and gives us an output  $Z$  ( $z = W_1X_1 + W_2X_2 + b$ ), which is then passed on to the activation function  $F$ .

The activation function converts the output into a particular range. In essence, it interprets the output  $Z$ .

Some activation functions are:

1 step function

2 ReLU

3 Tanh

## How does perceptron work?

Let's say we have student placement data -

IQ	CGPA	placed
78	7.8	1
69	5.1	0

Perceptron has two stages: training and prediction

in training we try to find the best values of weights and bias

We feed each student's data into our model and attempt to predict whether or not he will be placed. And, because we already know the outcome, we will make changes to our model based on prediction and actual output during training.

Now we have values of weights and bias that are  $W_1$   $W_2$  and  $b$

for prediction say we have a student with an IQ of 100 and with CGPA of 8 then -

$$Z = 100w_1 + 8w_2 + b$$

This  $Z$  will be sent to the activation function, and the activation function will decide whether this student will be placed or not.

Say we have a step-function as an activation function

Therefore if  $Z \geq 0$  output will be 1 and output will be 0 otherwise.

## How to train a perceptron?

Training a perceptron means finding the best values of weights and bias for given data.

Loss function: it tells us how good our predicted parameters are on given data. We can say the loss function calculates the error for our predicted values of parameters.

Let's say we have data in the form:

$X_1$	$X_2$	$y$
$X_{11}$	$X_{12}$	$y_1$

$X_{21}$	$X_{22}$	$y_2$
----------	----------	-------

Where  $X_1$  and  $X_2$  are features and  $y$  is our predicted values.

$$L = \frac{1}{n} \sum \max(0, -y_i f(x_i))$$

$$f(x_i) = W_1 X_{i1} + W_2 X_{i2} + b$$

We have to minimise the loss by changing the values of parameters that are weights and bias.

We use a gradient descent algorithm to find weights such that the loss is minimal.

## Gradient Descent -

The gradient descent algorithm returns the coordinate points where the value of a function is the smallest.

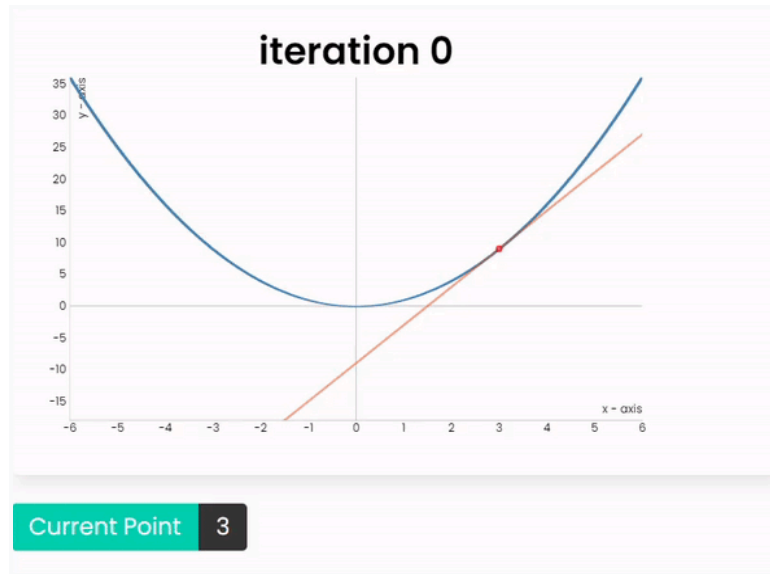
So if we apply a gradient descent algorithm to the loss function of a perceptron, it will give us a point (in this case,  $w_1$ ,  $w_2$ , and  $b$ ) for which the loss is minimum.

Let's say we have  $f(x) = x^2$  and we apply gradient descent to this function:

Our starting point is  $X = 3$  and we will update our point by following formula:

$$X_{new} = X_{old} + \eta \cdot \frac{\partial f}{\partial x}$$

Here I took learning rate( $\eta$ ) to be 0.2 then gradient descent on  $f(x) = x^2$  will look like this:

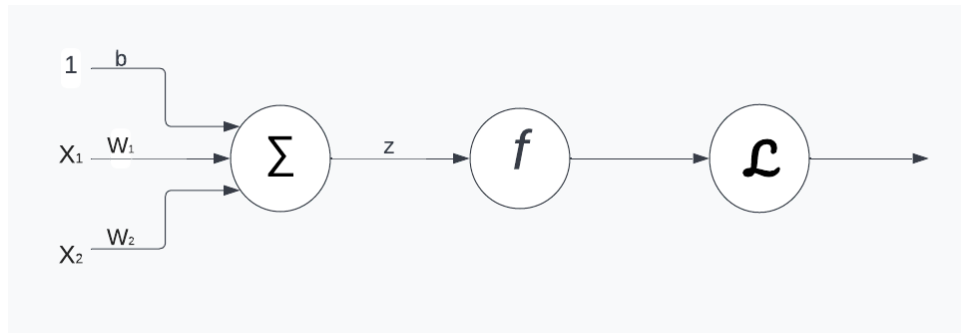


We can interpret this formula as  $df/dx$  determining the direction in which we should move and  $\eta$  telling us how much we should move in that direction. It is important to use  $\eta$  because if there's no  $\eta$  then we can pass the minima of  $f(x)$  and go to the other side of a function and can oscillate between 2 values of  $X$ .

In this case, I took a simple example, but the function can be composed of more than one variable. In that case, we have to update each variable using the same formula we used earlier, but instead of  $X$ , the new variable will be there. It is an iterative algorithm, so we have to do this over and over.

For perceptron gradient descent will be applied as follows -

Perceptron is a very flexible model; if we change its loss function and activation function, we will get different outputs.



Loss Function( $\mathcal{L}$ )	Activation function	Output
Hinge Loss	Step function	perceptron - binary classifier
log-loss(binary cross entropy)	sigmoid	Logistic regression - binary classifier
Categorical cross entropy	softmax	softmax regression - multiclass classifier
MSE(mean square error)	Linear(No activation function)	Linear regression

## Problem with. Perceptron-

A perceptron can't be fitted to data that is not linearly separable.

This means if one line, plane, or hyperplane cannot separate your data, then we can't fit Perceptron on it.

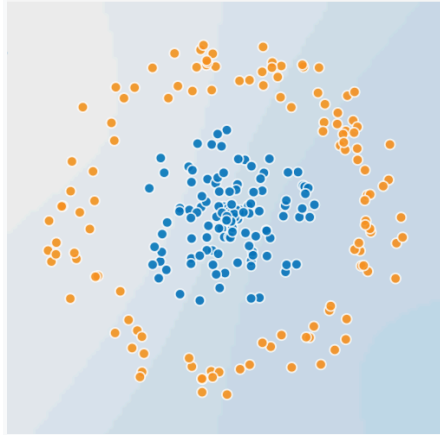


Fig.1

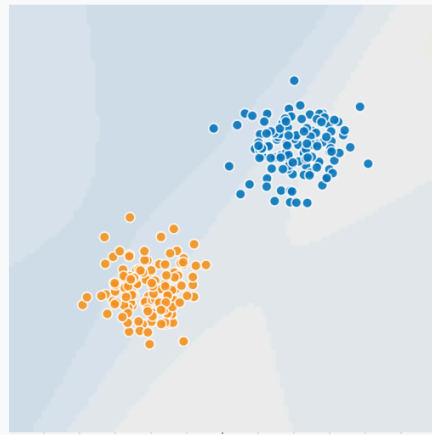


fig.2

Because in Fig. 1 the line cannot separate the orange dots and blue dots, the perceptron model cannot be fitted.

In fig. 2, we can separate orange dots and blue dots using a single line. Therefore, in this case, we can use perceptron for classification.

The [Tensorflow Playground](#) is a very good GUI tool to visualise this.

## Multilayer perceptron

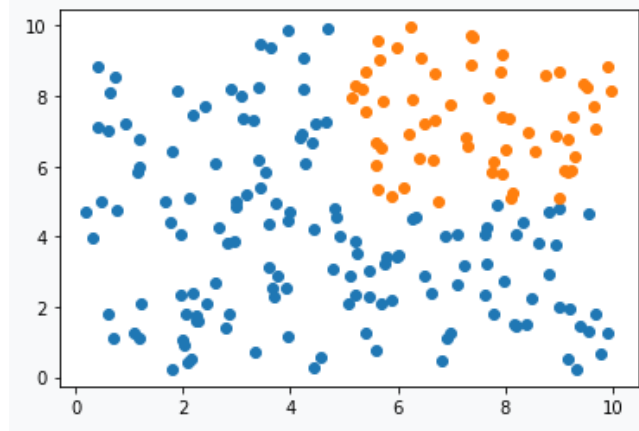
### MLP intuition

How does MLP capture non-linearity?

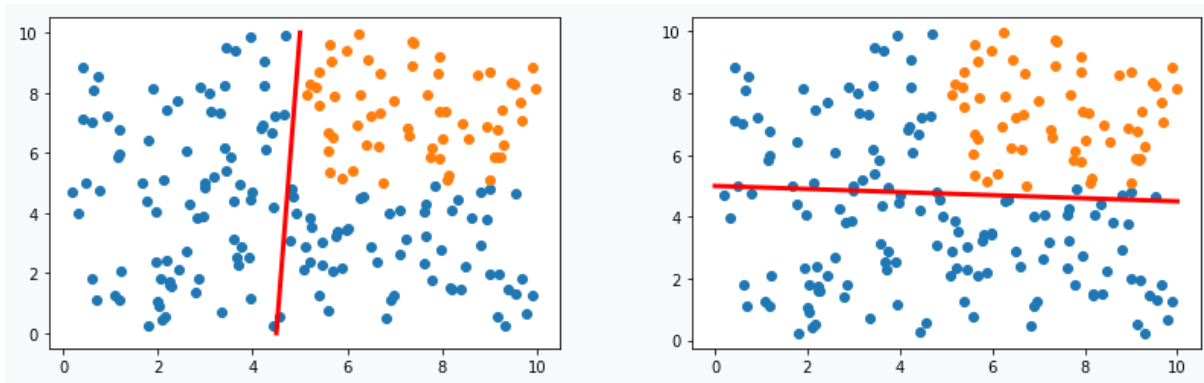
We will consider perceptrons with sigmoid as the activation function and log-loss as the loss function.

Sigmoid gives us the probability of something.

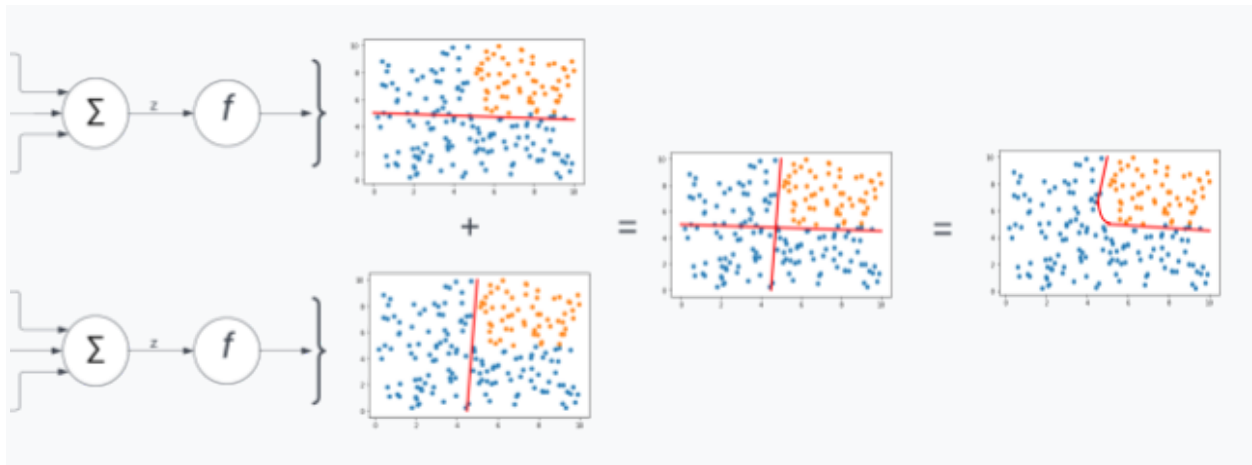
Let's say we have non-linear data.



Say we trained 2 perceptrons on the above data and got the following decision boundary:



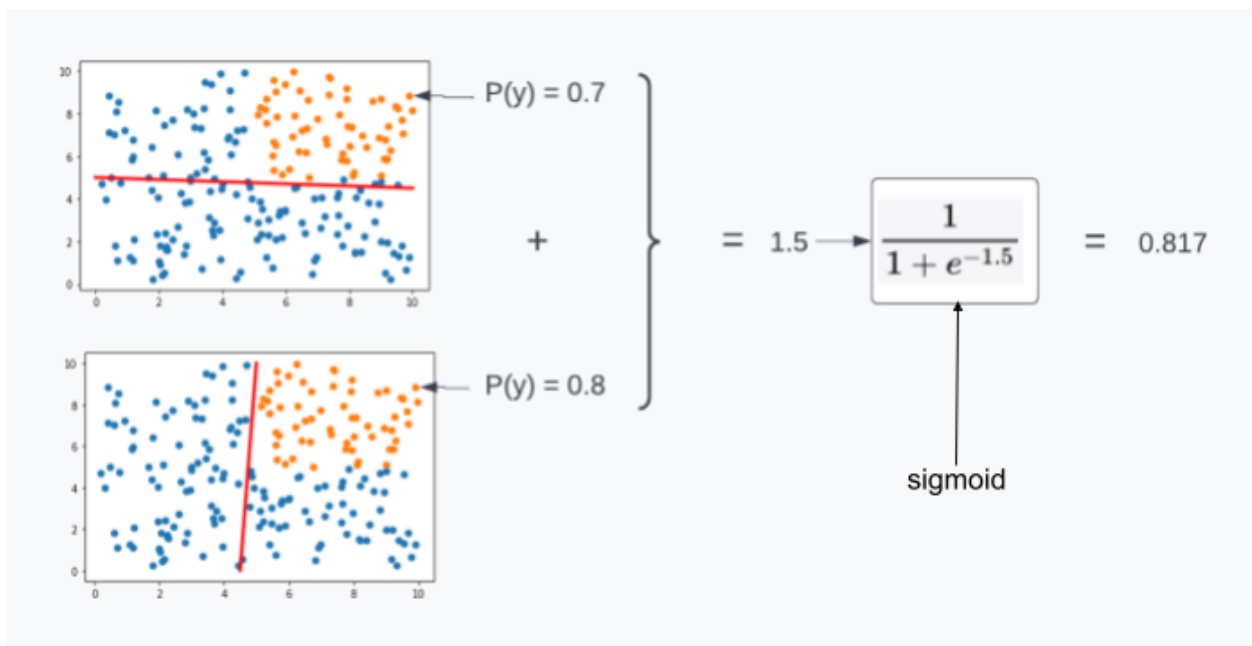
→ Now, if we superimpose these boundaries, we will get our desired boundary.



→ This is a very informal, non-mathematical way of showing how MLP captures the non-linearity.

Let's see the math behind it.

Let's say we have 2 perceptrons:



So we chose a single point and added the probabilities of that point from both perceptrons, and then sent



the output to the sigmoid function to produce a result between 0 and 1.

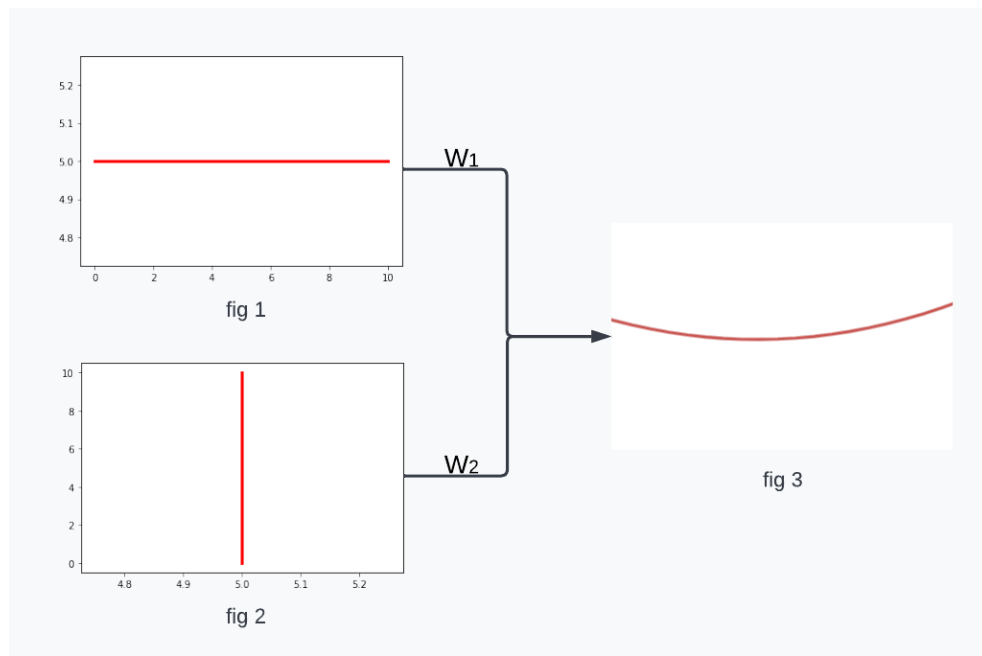
Do this process for every point and we will get our desired decision boundary.

We can say we took a linear combination of both perceptrons.

We will follow the same procedure for all points.

By doing this, we will get our desired decision boundary.

I say linear combination, not addition, because:



Say we have 2 perceptrons, fig 1 and fig 2, and we want a decision boundary like in fig 3.

We can see that to get the decision boundary in fig 3, the decision boundary in fig 1 will have more effect (weight) than the decision boundary in fig 2.

And that's where the concept of weights comes in.

$$W_1 > W_2$$

Inputs in preceptron 3 will be  $w_1 \cdot \text{output 1}$  and  $w_2 \cdot \text{output 2}$ .

Therefore, it is a linear combination.

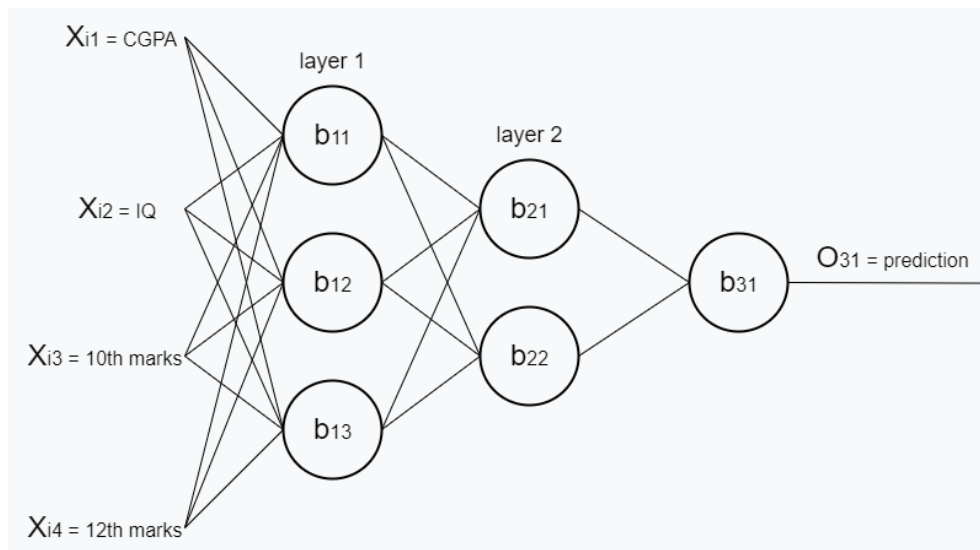
Now we can see that the outputs of 2 perceptrons are serving as an input to the 3rd perceptron.

Note: After each node, the activation function (in this case, sigmoid) is applied to the output.

How does MLP/ANN make the prediction?

(Forward propagation)

Let's say we trained our model and got all the trainable parameters, i.e. weights and bias.



The trainable parameters are  $12 + 3 + 6 + 2 + 1 = 26$ .

Say we want to predict whether a student will get placed or not based on his/her CGPA, IQ, 10th marks

and 12th marks.

CGPA	IQ	10th percentage	12th percentage	placed
7.2	72	69	81	1
8.1	92	75	76	0

$$\text{prediction} = \sigma(W^T X + b)$$

Layer #1

$$\underbrace{\begin{bmatrix} W_{11}^1 & W_{12}^1 & W_{13}^1 \\ W_{21}^1 & W_{22}^1 & W_{23}^1 \\ W_{31}^1 & W_{32}^1 & W_{33}^1 \\ W_{41}^1 & W_{42}^1 & W_{43}^1 \end{bmatrix}^T}_{3 \times 4} \underbrace{\begin{bmatrix} X_{i1} \\ X_{i2} \\ X_{i3} \\ X_{i4} \end{bmatrix}}_{4 \times 1} + \underbrace{\begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}}_{3 \times 1}$$

→ Here if we insert the 1st row-

$$X_{i1} = 7.2, X_{i2} = 72, X_{i3} = 69, X_{i4} = 81$$

→We have to find the sigmoid of the resultant matrix.

$$\sigma \left( \begin{bmatrix} W_{11}^1 X_{i1} + W_{21}^1 X_{i2} + W_{31}^1 X_{i3} + W_{41}^1 X_{i4} + b_{11} \\ W_{12}^1 X_{i1} + W_{22}^1 X_{i2} + W_{32}^1 X_{i3} + W_{42}^1 X_{i4} + b_{12} \\ W_{13}^1 X_{i1} + W_{23}^1 X_{i2} + W_{33}^1 X_{i3} + W_{43}^1 X_{i4} + b_{13} \end{bmatrix} \right)$$

$$= \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}$$

Layer #2

$$\underbrace{\begin{bmatrix} W_{11}^2 & W_{12}^2 \\ W_{21}^2 & W_{22}^2 \\ W_{31}^2 & W_{32}^2 \end{bmatrix}}_{2 \times 3} \underbrace{\begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}}_{3 \times 1} + \underbrace{\begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}}_{2 \times 1}$$

$$= \sigma \left( \begin{bmatrix} W_{11}^2 O_{11} + W_{21}^2 O_{12} + W_{31}^2 O_{13} + b_{21} \\ W_{12}^2 O_{11} + W_{22}^2 O_{12} + W_{32}^2 O_{13} + b_{22} \end{bmatrix} \right)$$

$$= \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix}$$

Layer #3

$$\underbrace{\begin{bmatrix} W_{11}^3 \\ W_{21}^3 \end{bmatrix}}_{1 \times 2}^T \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + \underbrace{[b_{31}]}_{1 \times 1}$$
$$= \sigma([W_{11}^3 O_{21} + W_{21}^3 O_{22} + b_{31}])$$
$$= \hat{y} = O_{31} = \text{prediction}$$

\* How to train MLP/ANN?

(back propagation)

→ Let's say we have a regression problem.

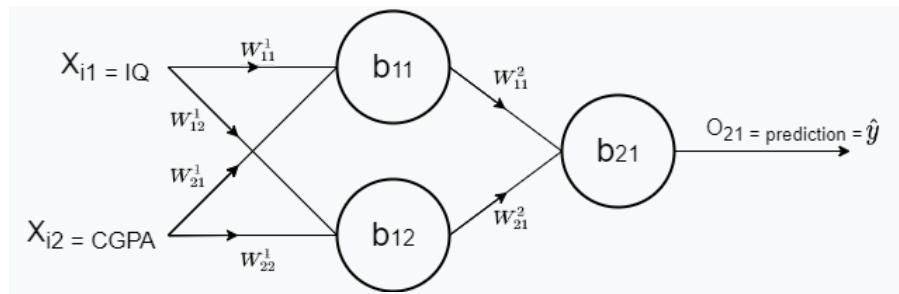
→ activation function will be linear.

Let's say we have to predict LPA based on the IQ and CGPA of a student.

IQ	CGPA	LPA
80	8	3
60	9	5

70	5	8
120	7	11

Let's say we have the following neural network architecture to do that.



$4+2+3 = 9$  trainable parameters.

step 0-

Initialize the weights and bias.

Let's say  $W \rightarrow 1, b \rightarrow 0$

step 1

choose 1 student i.e. 1 row From data and feed his/her data to the neural network. and try to predict lpa.

step 2-

Choose a loss function.

In this case I take MSE loss because it is a regression problem.

$$\text{MSE Error} = L = (y - \hat{y})^2$$

$y \rightarrow$  true value of LPA present in dataset

$\hat{y} \rightarrow$  predicted value we got by forward propagation

Calculate the loss for that 1 student.

step 3-

Update weights and bias such that it reduces the loss. We will use gradient descent to do this.

$$W_{new} = W_{old} - \eta \cdot \frac{\partial L}{\partial W_{old}}$$

$\rightarrow$  Our neural network have 9 trainable parameters

Therefore we have to update 9 Parameters every time.

$$b_{new} = b_{old} - \eta \cdot \frac{\partial L}{\partial b_{old}}$$

Now we have to find-

$$\begin{aligned}\frac{\partial L}{\partial W_{12}^1} &= -2(y - \hat{y}) \cdot W_{21}^2 \cdot X_{i1} \\ \frac{\partial L}{\partial W_{22}^1} &= -2(y - \hat{y}) \cdot W_{21}^2 \cdot X_{i2} \\ \frac{\partial L}{\partial b_{12}} &= -2(y - \hat{y}) \cdot W_{21}^2\end{aligned}$$

## Recurrent Neural Network (RNN) -

It is a special type of neural network.

It is a sequential model and is used to work on sequential data.

Sequential data means data where sequence matters.

E.g. -

- i) text → sequence matters.
- ii) time series data.

In ANN, we feed all the information at once, therefore important sequential information is lost.

ANN does not perform well on sequential data.

RNN is being heavily used in natural language processing.

Textual data can be of different sizes, and we can't vary the input size in ANN; that's why we use an RNN on textual data.

### Applications of RNN-

- sentiment analysis
- Sentence completion (Gmail).



- Image caption generation.
- Google translate.

## Data for RNN-

RNN requests input data in a very specific format.

Let's say we have the following dataset:

	Reviews (X)	Sentiment
$X_1$	Movie was good	1
$X_2$	Movie was bad	0
$X_3$	Movie was not good	0

We feed the data to RNN in the following form:

(time steps, input features)

First, let's convert words using one hot encoding.

Our vocabulary has 5 words.

movie = [1,0,0,0,0]  
Was = [0,1,0,0,0]  
Good = [0,0,1,0,0]  
Bad = [0,0,0,1,0]  
Not = [0,0,0,0,1]

Therefore, review 1, i.e., the movie was good, can be represented as:

$[[1,0,0,0,0], [0,1,0,0,0], [0,0,1,0,0]]$

In RNN, we feed data word by word.

Therefore if we feed the 1st word of 1st review then time step = 1

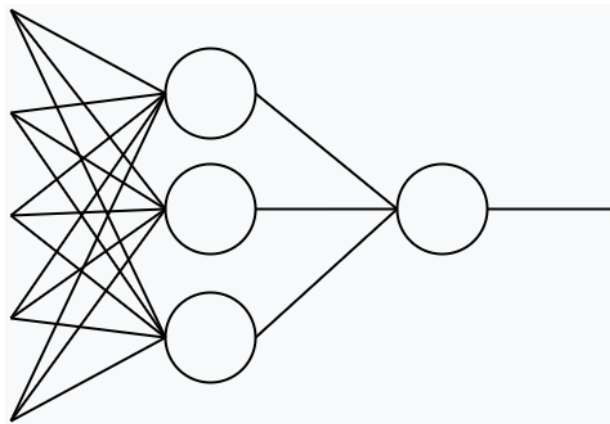
i.e.  $(1, [1,0,0,0,0])$

Do this for each word in a given review before moving on to the next.

### How RNN works?-

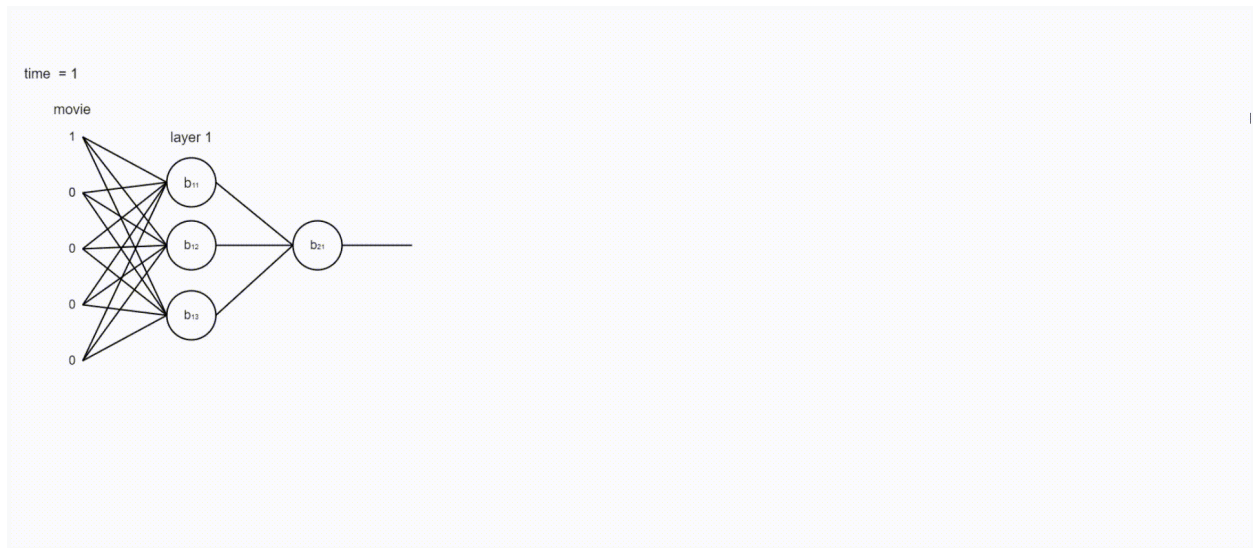
Let's take the same example we gave on the last page.

Let's say we are doing sentiment analysis and we have the following RNN architecture:



In an RNN, we send input word by word, and in the given example, we represent each word with a 5-dimensional vector. Therefore, 5 inputs in the input layer make sense.

Let's say we pass 1st review  $X_1$  to the model.



1st we inputted the 1st word of 1st review i.e.

$X_{11}$  = movie (time=1).

We calculated the output of the 1st layer as we calculate output in ANN by doing dot products with weights, adding bias, and applying the activation function.

Now we input the 2nd word, i.e.  $X_{12}$  = "was" (time = 2)

But in addition to the 5-dimensional vector, we will input the output of layer 1 we calculated in (time = 1).

These additional inputs ( $O_{11}$ ,  $O_{12}$ , and  $O_{13}$ ) will have their own weights.

We made no changes to the weights or bias in (time = 2). It is the same as (time = 1).

Now we calculate the output again and repeat these steps (i.e., time = 3, input  $X_{13}$  = "good" along with output [ $O'_{11}$ ,  $O'_{12}$ ,  $O'_{13}$ ]).

Now we have exhausted all the words in the 1st review.

Now just take the dot product with weights i.e.

$$[O''_{11}, O''_{12}, O''_{13}] \cdot [W_{11}^2, W_{21}^2, W_{31}^2]$$

= output

and  $\sigma(\text{output}) = \text{final prediction}$

Note: To make the model consistent, we add 3 additional inputs and their weights in layer 1, but we randomly initiate their values.

Let's see if math add up or not

$W_i \rightarrow$  original weight matrix (5X3)

$W_h \rightarrow$  weight matrix for additional inputs (3X3)

$O_1 \rightarrow$  output of layer 1 at time = 1 (1X3)

$b_1 \rightarrow$  bias matrix of layer 1 (1X3)

$W_i \rightarrow$  original weight matrix (5,3)

$W_h$  - weight matrix for additional inputs (3,3).  
(observation this matrix always be square matrix)

$O_1$  - output of layer 1 at time = 1 (1,3)

$b_1$  - bias matrix of layer 1 (1,3).

@time=1 # layer 1

$$f(X_{11}W_i + O_0W_h^o + b_1) = O_1$$

$X_{11} \rightarrow$  1st word of 1st review

$O_0$  and  $W_h^0 \rightarrow$  randomly initiated values

Let's check if the dimensions match up or not:

$$\begin{aligned} &= (1, 5) \cdot (5, 3) + (1, 3) \cdot (3, 3) + (1, 3) \\ &= (1, 3) + (1, 3) + (1, 3) \\ &= (1, 3) \\ &= O_1 \end{aligned}$$

This is correct; our output should be of dimension (1,3).

@time=2 # layer 1

$$f(X_{12}W_i + O_1W_h^o + b_1) = O'_1$$

$X_{12} \rightarrow 2^{nd}$  word of  $1^{st}$  review  
 $W_i, W_h^o, b_1 \rightarrow$  unchanged, same as last time step  
 $O_1 \rightarrow$  output from previous time step

@time=3 #layer 1

$$f(X_{13}W_i + O'_1W_h^o + b_1) = O''_1$$

$X_{13} \rightarrow 3^{rd}$  (and last) word of  $1^{st}$  review  
 $W_i, W_h^o, b_1 \rightarrow$  same as last time step  
 $O'_1 \rightarrow$  output of previous time step

We have used all the words from the first review.

Now we take  $O''_1$  to the next layer

$$f(O''_1 \cdot W_{11}^2 + b_2) = \text{output} = \hat{y}$$

Let's check if we got a single value as an output:

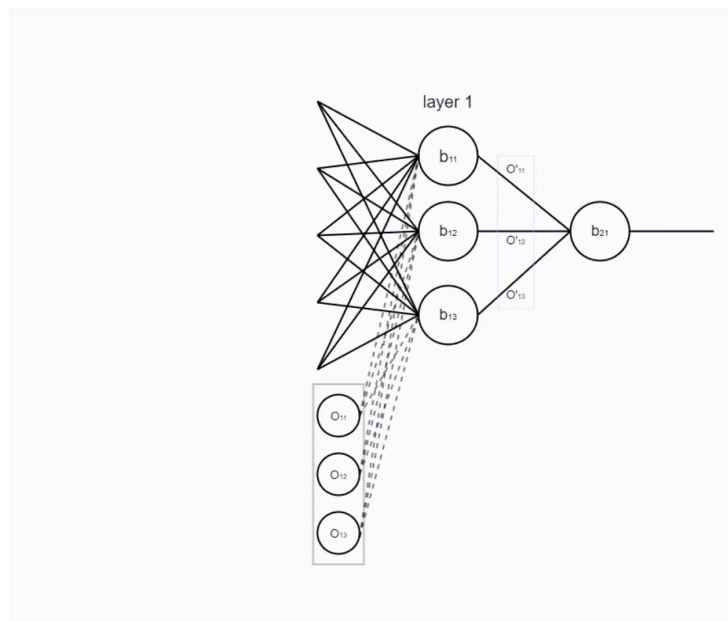
$$(1, 3) \cdot (3, 1) = 1$$

And that is how we do prediction in RNN. (Forward propagation)

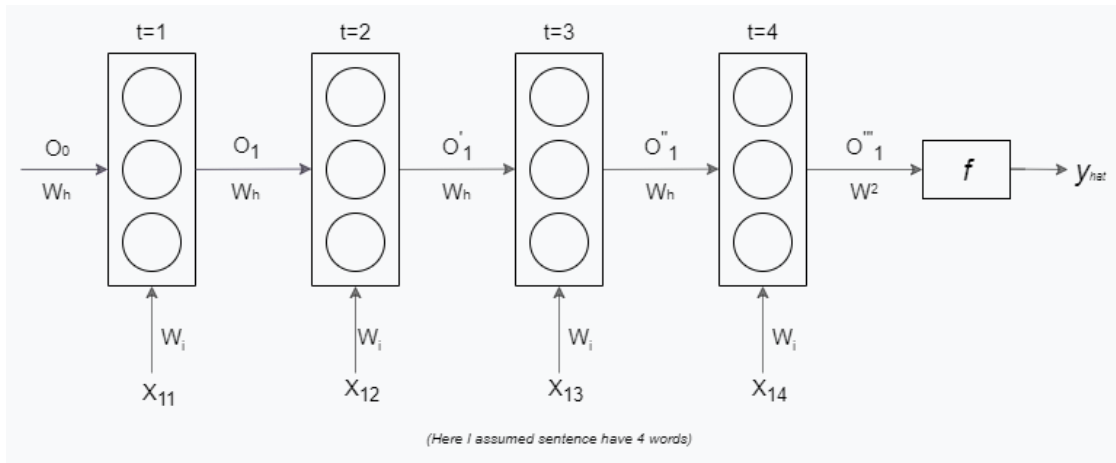
Note: Here, there is only 1 RNN layer; if there is more than 1 RNN layer, each layer will have additional inputs and weights associated with it.

## How to train RNN?

Let's say we have the following RNN -



Which is equivalent to:



In this RNN total trainable parameters are-

$$(3 \times 5) + (3 \times 1) + (3 \times 3) + 4 = 31$$

We will use the same method to update weights and bias as ANN

Step 1 - 1st initialize  $O_0$  and  $W_h$

Step 2 - Select row from data and do forward propagation

Step 3 - Calculate loss

Let -

$$\text{Loss} = (\hat{y} - y)$$

$\hat{y} \rightarrow$  predicted value

$y \rightarrow$  original value

Step 4 - Use gradient descent to update weights and bias.

$$O_1 = f(X_{11}W_i + O_0W_h^0 + b_1)$$

$$O'_1 = f(X_{12}W_i + O_1W_h^0 + b_1)$$

$$O''_1 = f(X_{13}W_i + O'_1W_h^0 + b_1)$$

$$O'''_1 = f(X_{14}W_i + O''_1W_h^0 + b_1)$$

$$\hat{y} = f(O'''_1 \cdot W^2 + b_2)$$

$$W_{new}^2 = W^2 - \eta \cdot \frac{\partial L}{\partial W^2}$$

$$W_{i_{new}} = W_i - \eta \cdot \frac{\partial L}{\partial W_i}$$

Same for bias

Step 5 - Go to step 2.

## BERT

BERT stands for Bidirectional Encoder Representations from Transformers.

BERT is a tool that understands human language better than any other tool we have had.

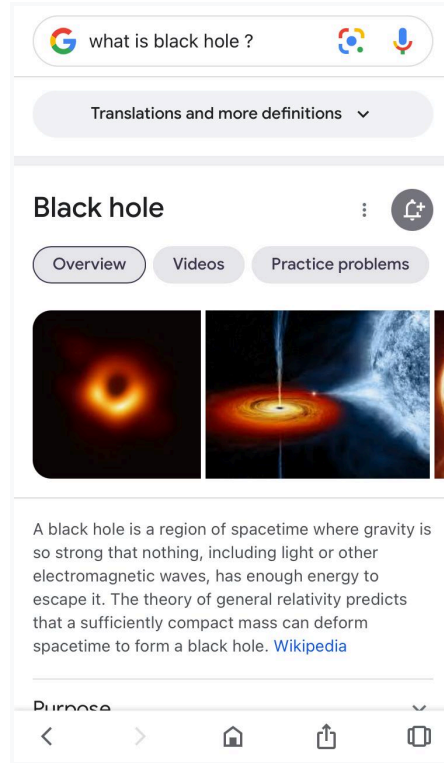
BERT is trained on English Wikipedia and the concatenation of book corpora.

BERT is based on transformer architecture.

### Applications of BERT -

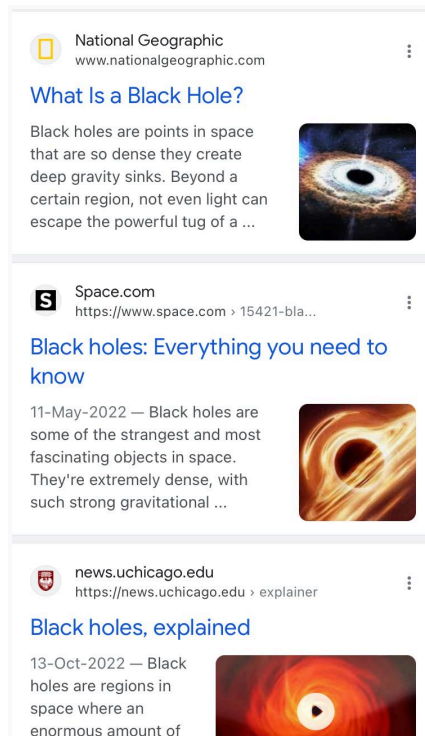
1. Text summarization -





On Google, I searched the question "What is a black hole?" and as we can see, it gave me a summary of a Wikipedia article about the black hole.

## 2. Text encoding similarity retrieval -



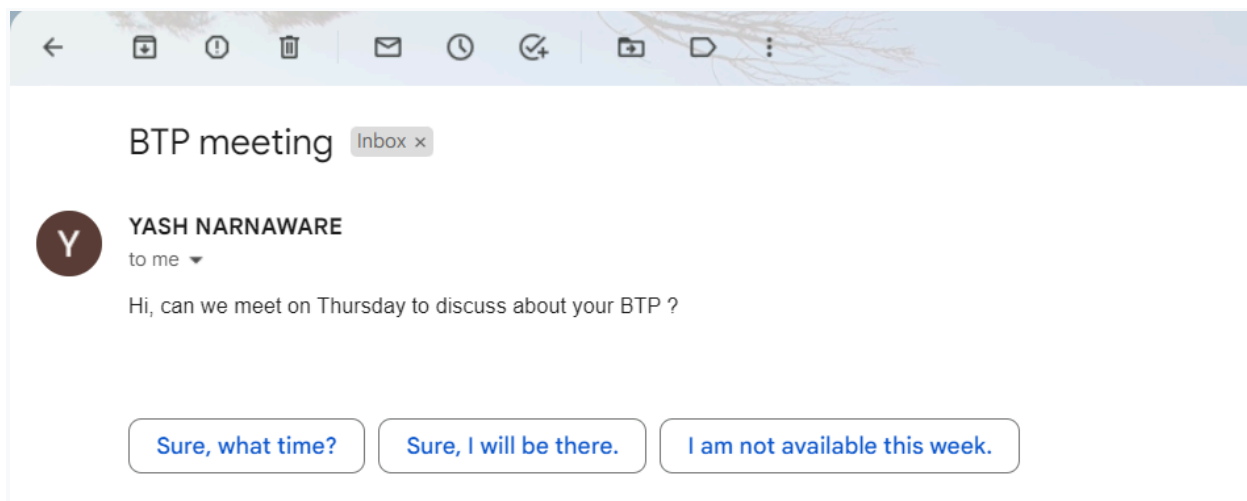
Here I searched for the same question, which is "What is a black hole?" And scroll down, and as we can see, Google gave us the related articles to the question.

### 3. Question answering -



I asked Google, "Who is CEO of tesla?" and it gave me Elon Musk as an answer.

### 4. Response selection -



When we get an email on Gmail, we see these responses, in this case, "Sure, what time?" or "Sure, I will be there." etc.

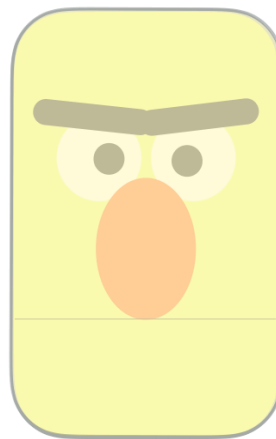
These are some of the applications of BERT.

When we input a sentence into a BERT, we get word embeddings for each word and the whole sentence as well.

Let's look at what's inside the BERT.

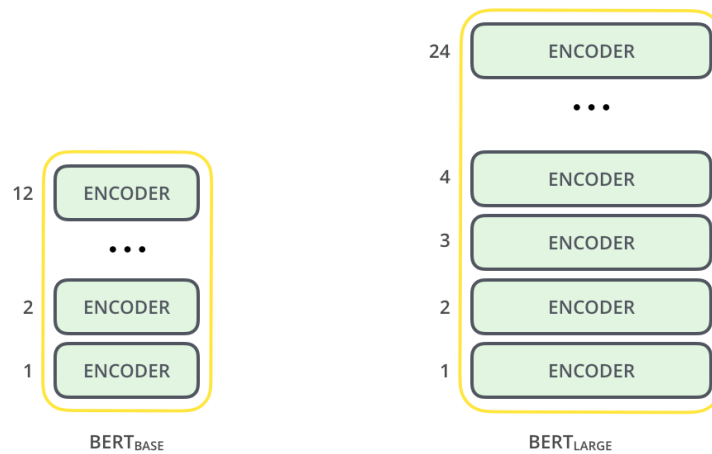


BERT<sub>BASE</sub>



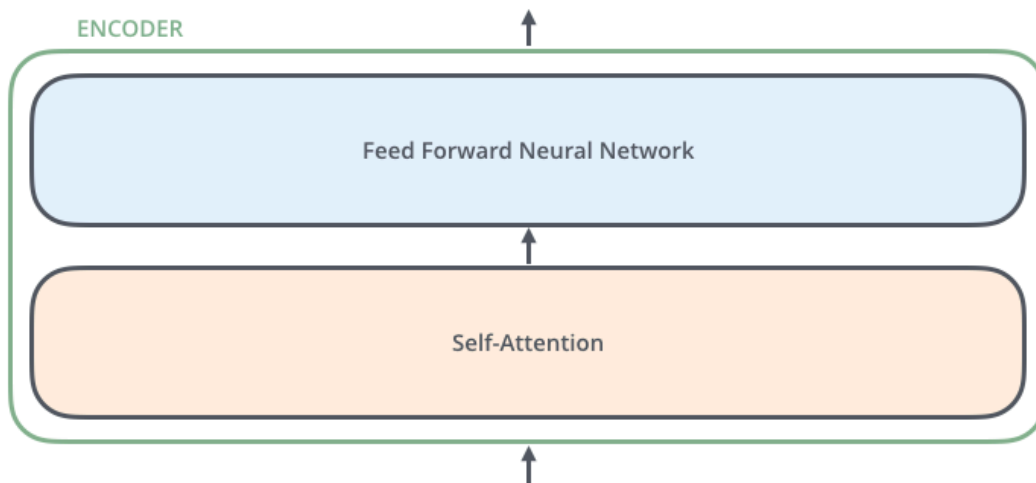
BERT<sub>LARGE</sub>

There are 2 types of BERT: BERT base and BERT large.

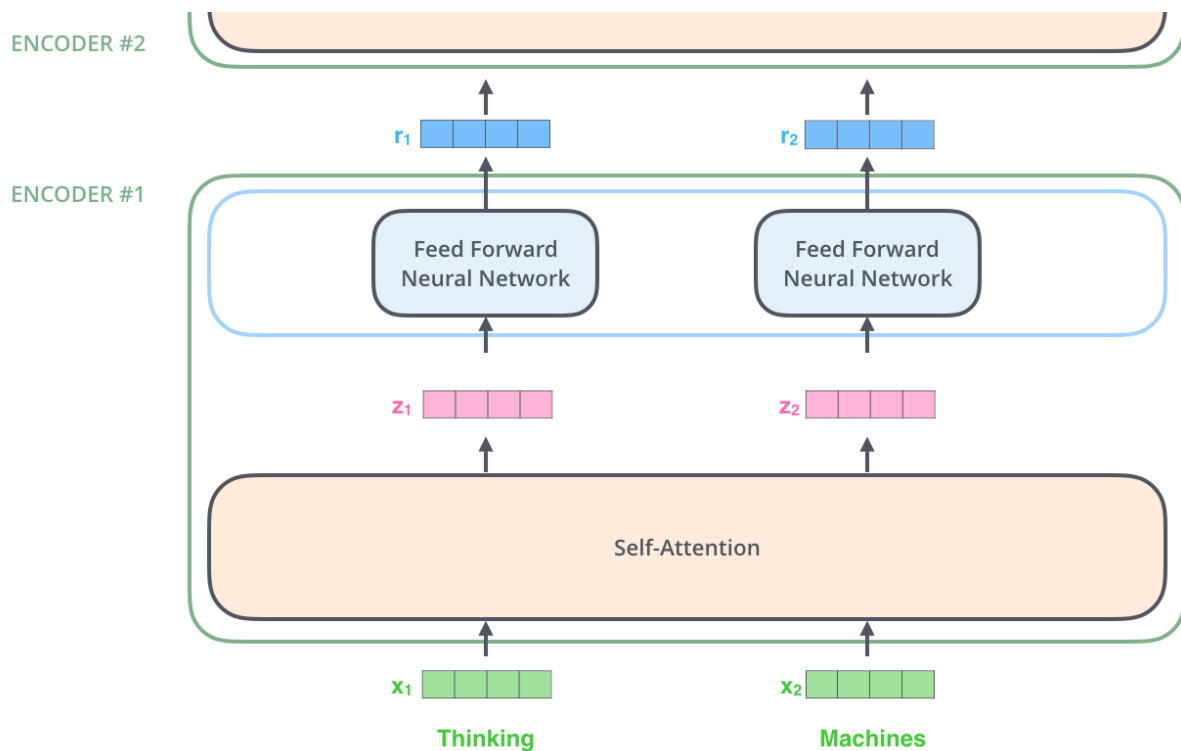


The difference between BERT base and BERT large is that BERT base has 12 encoders and BERT large has 24 encoders.

Now let's look at what's inside an encoder.



Inside the encoder, there is self-attention and a feed-forward neural network, which is just a normal artificial neural network (ANN).



So let's say we are passing 2 words "thinking" and "machines" to the BERT, then they will go through self-attention, and the output of self-attention will be an input to a feed-forward neural network. The output of 1 encoder will act as input for the next encoder.

We already know what a feed-forward neural network is. Let us define self-attention:

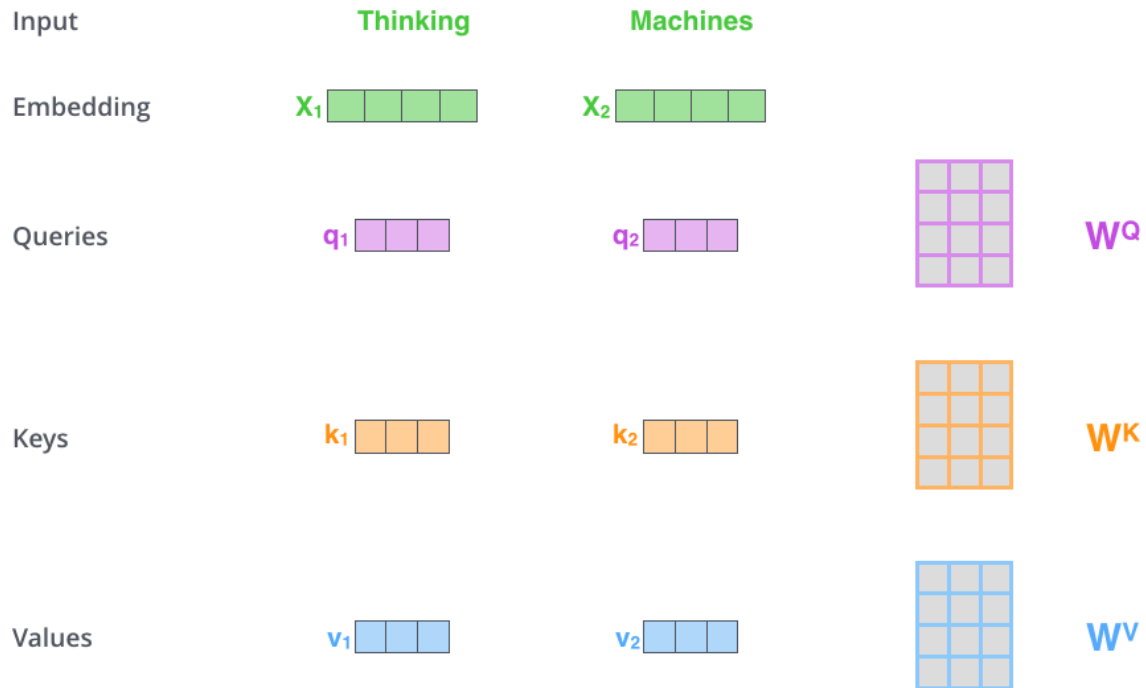
"The animal didn't cross the street because it was too tired"

Let's say we have the above sentence. For humans, it is very easy to determine what the word "it" is referring to, but for machines, it is not that clear. Machines will get confused about whether "it" is referring to the word "animal" or "street".

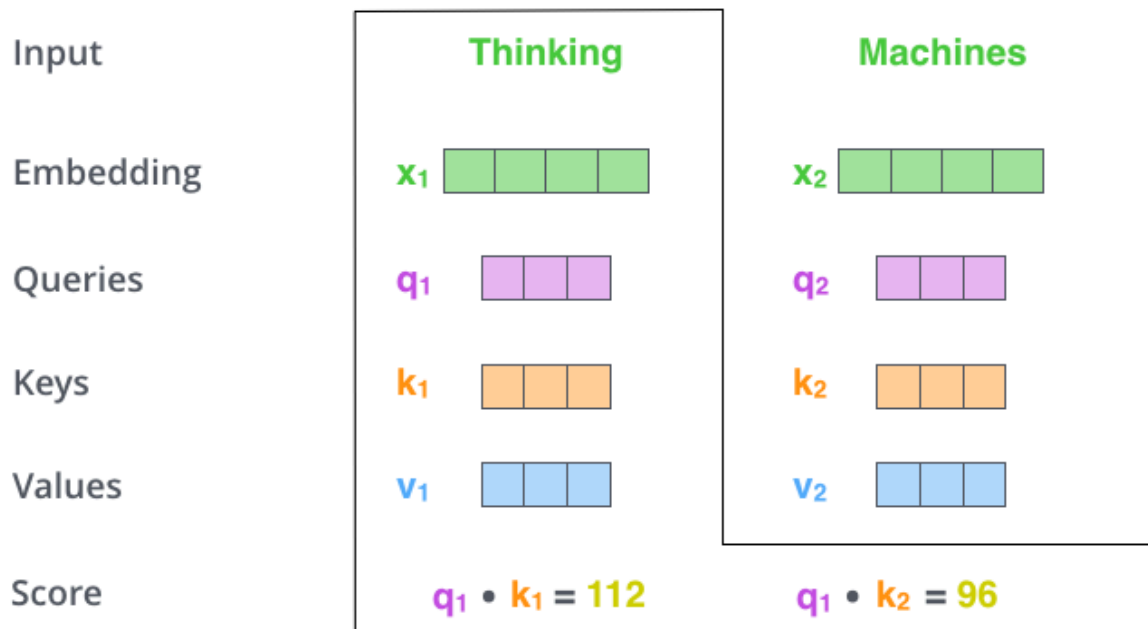
When we pass this sentence through self-attention, then machines will also understand that the word "it" is referring to the word "animal".

Now let's see how self-attention works:

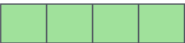
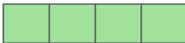
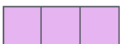

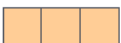
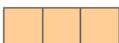
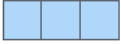
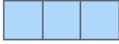
Again, we are using just 2 words for simplicity.



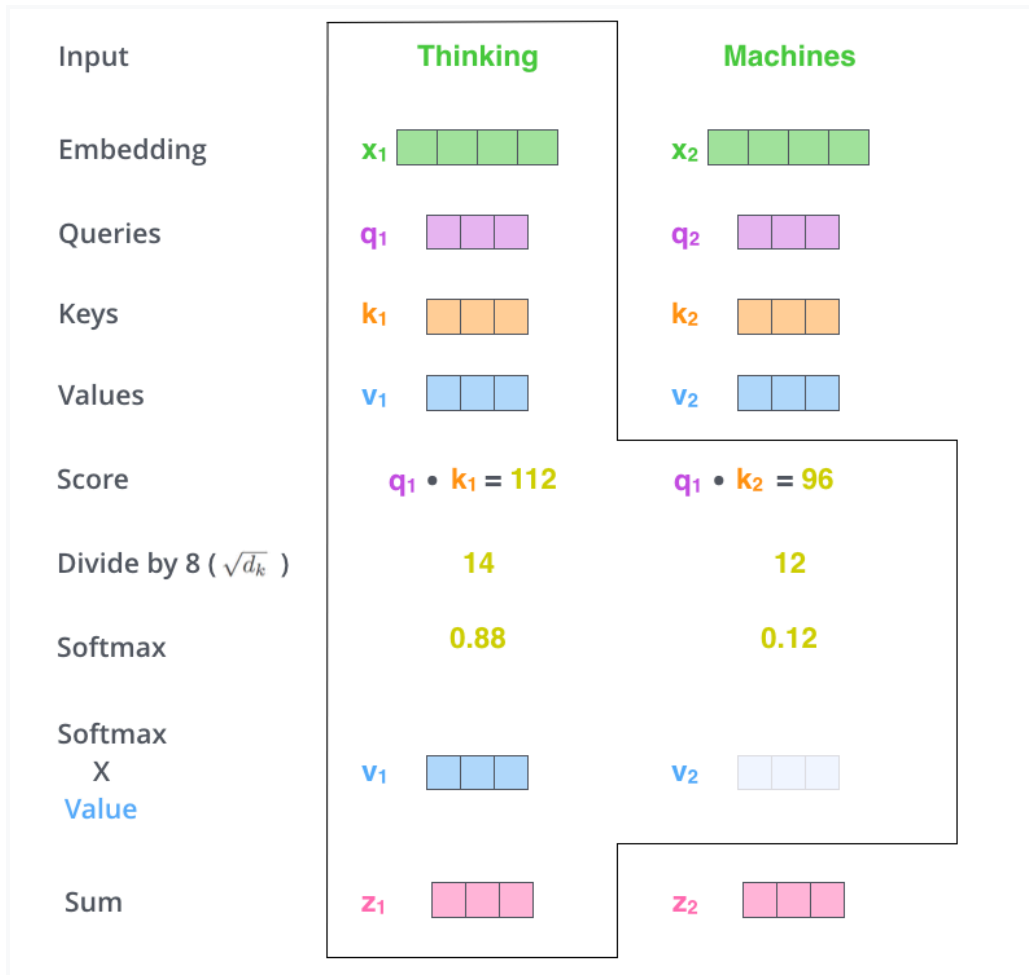
We have 2 words and their embeddings (Word2Vec, I guess), and then we have 3 matrices:  $W^Q$  (query matrix),  $W^K$  (keys matrix), and  $W^V$  (value matrix). These matrix values are obtained during the training of our model. We multiply each of these matrices with our word embedding vectors to get the query vector, keys vector, and values vector for each word.



Then we calculate a score for each word with respect to another word. Scores can be calculated by taking the dot product of that word's query vector and all of the key vectors.

Input	Thinking	Machines
Embedding	$x_1$ 	$x_2$ 
Queries	$q_1$ 	$q_2$ 
Keys	$k_1$ 	$k_2$ 
Values	$v_1$ 	$v_2$ 
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ( $\sqrt{d_k}$ )	14	12
Softmax	0.88	0.12

Then we divide the score by 8 (the square root of the dimension of the keys vector). Then we apply the softmax function to the resultant values.



Then we multiply the softmax score with the value vector of a word and sum those vectors to get output, which will be passed on to the feed-forward neural network.

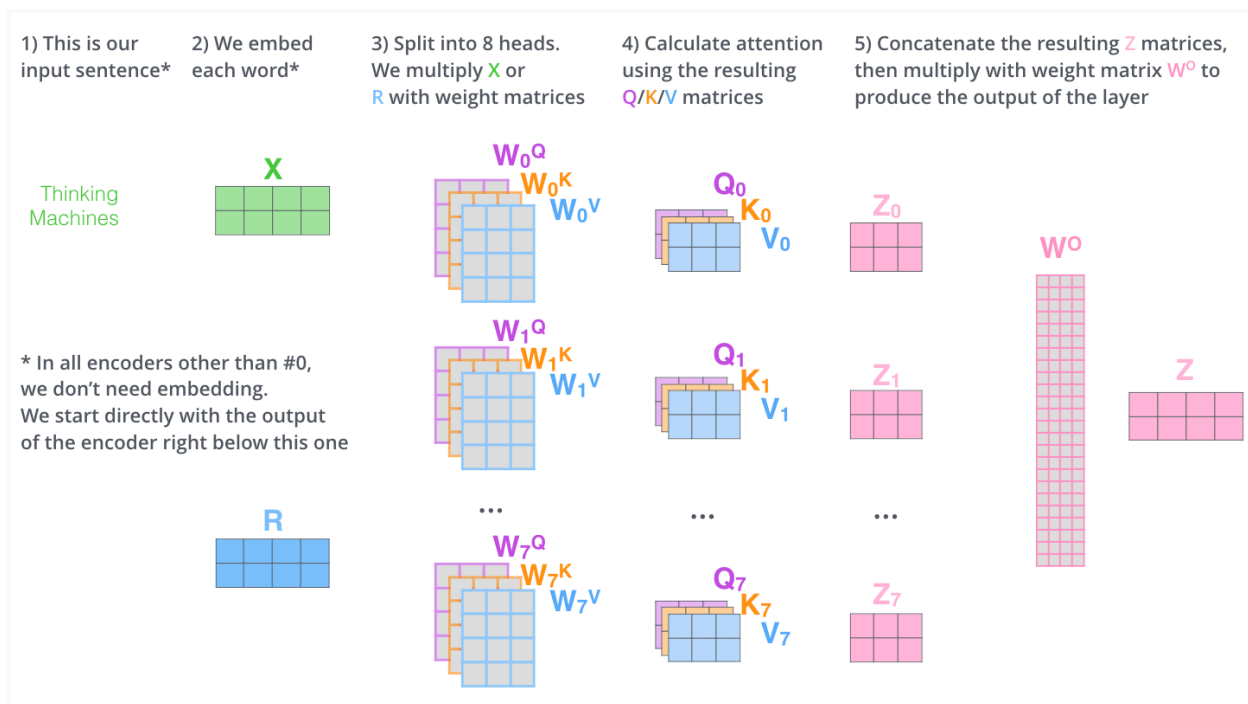
All the steps we have taken can be expressed using this formula:



$$\text{softmax}\left(\frac{\overset{\text{Q}}{\begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix}} \times \overset{\text{K}^T}{\begin{matrix} \square & \square \\ \square & \square \\ \square & \square \end{matrix}}}{\sqrt{d_k}}\right) \overset{\text{V}}{\begin{matrix} \square & \square \\ \square & \square \end{matrix}}$$

$$= \overset{\text{Z}}{\begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix}}$$

This is only one attention head. BERT has 8 attention heads, which means it has 8 sets of query, key, and value matrices, and for each word, we will get 8 vectors as output.



We will concatenate those 8 vectors and multiply the result by the matrix  $W^O$ , whose values we will get after training the model. And this will be the output of the attention layer.

